开篇词-软件设计，应对需求规模的“算法”

你好，我是郑晔！

作为一个能把基本功能实现出来的程序员，偶尔仰望天空时，你是否这样问过自己，“我写过的代码还有没有更好的写法呢？如果有一天，我能够把它重写一遍，我该怎么做呢？”

这就是我当年问过自己的问题，因为我对自己的代码不满意：

- 我厌倦了，把各种代码堆砌在一起，然后，在出现Bug时，犹如“大家来找茬”一样在其中定位问题；
- 我厌倦了，仅仅为了一个小需求，要在无数的地方小心翼翼地做着各种微调，还被产品经理嫌弃改得慢；
- 我厌倦了，自己辛辛苦苦写好的代码，被别人在其他地方不经意地修改，给弄崩溃了；
- ……

我四处寻找答案，直到后来，我找到了一个东西，叫做“软件设计”。在如饥似渴地学习了软件设计之后，我对做软件这件事有了全新的认识：

- 我知道了，写软件不仅要追求如何实现功能，还要考虑未来的维护和扩展；
- 我知道了，代码不应该毫无目的地堆在那里，而是要考虑如何组织更为恰当；
- 我知道了，原来后期遇到很多问题，只是因为前期缺乏设计而留下了隐患；
- ……

如果你也曾有同样的迷茫，想破茧成蝶，欢迎你与我一起探索“软件设计”！

关注长期变化的软件设计

> 设计是为了让软件在长期更容易适应变化。

> Design is there to enable you to keep changing the software easily in the long term.

> —— Kent Beck

软件设计，是一门关注长期变化的学问。它并不是程序员的入门第一课，因为初窥编程门径的程序员首先追求的是把一个功能实现出来，他无法看到一个软件长期的变化。

或许未曾学过软件设计，但这并不妨碍你凭借一股蛮力把软件写出来。但如果你在做的是一个有生命力的软件，长期意味着什么呢？意味着会有源源不断的需求扑面而来。面对一拨又一拨的需求，你该如何应对呢？

你可以想象一下，小米加步枪的解决方案，在敌人不多的时候，你还可以应付得游刃有余，但当敌人已经漫山遍野时，你需要的是大规模杀伤性武器。而软件设计就是这个大规模杀伤性武器。

用程序员们更熟悉的排序算法为例，快速排序的平均复杂度是O(nlogn)，而插入排序是O(n^2)。所以，一般我们说快速排序比插入排序有优势。但是，这种优势只有在一定规模下才能体现出来。当数据规模很小的时候，二者差别并不明显，更有甚者，插入排序在某些情况下表现得会更好。但当数据规模很大时，快速排

序的优势就非常明显了。对比两个算法的优劣，关键在于数据规模。

所以你会发现，算法和软件设计其实是一样的，二者对抗的都是规模问题，只不过，**算法对抗的是数据的规模，而软件设计对抗的是需求的规模**。

你现在应该理解了，为什么软件设计是一门关注长期的学问了，因为只有长期的积累，需求才会累积，规模问题才会凸显出来。**软件设计，实际上就是应对需求的“算法”**。

如何学习软件设计？

软件设计的相关知识有很多，你可能听说过一些，比如，设计模式、领域驱动设计等等。但是，分别学习这些知识时，总有一些令人困惑的地方。比方说，学了那么多设计模式，你能用上的却没有几个；领域驱动设计中的概念那么多，你都不知道该从哪学起。

你的困惑我也有过，我花了很长时间才知道，我们困惑的，并不是这些知识本身，而是在于缺乏一个整体结构将它们贯穿起来。比如，多态，这个词在你看来只是一个普通的语法规则，但只有你懂得，它是把变的部分和不变的部分隔离开来，你才能理解开放封闭、接口隔离和依赖倒置这些设计原则的价值所在，理解了这些原则，才能知道某些设计模式只是这些原则在具体场景下的应用。

也就是说，**软件设计学习的难度，不在于一招一式，而在于融会贯通。**

所以，在这个课程中，我会尝试将软件设计的相关知识贯通起来，帮你建立起对软件设计的整体认知。

那具体要从哪里入手呢？其实对于“软件设计”，我们可以将其划分为两个维度：**“了解现有软件的设计”和“自己设计一个软件”。**

了解现有软件的设计，是能够在这个软件上继续添砖加瓦的前提。事实上，无论是初入职场，还是加入一个新公司，在工作初期，我们能做的往往也只是添砖加瓦，这时候如果能快速了解现有软件的设计，就可以尽快投入工作中去。此外，当你想从一个开源项目上汲取养分时，了解其背后的设计，也是一种不可或缺的能力。

大多数人在理解一个软件时，总会出现一个问题，就是眉毛胡子一把抓，直奔代码细节而去。这样不仅增加了我们的时间成本，还会迷失在细节之中，只见树木不见森林。所以在这个课程中，我会教你一个快速了解现有软件设计的方法，那就是抓住这个软件最核心的三个部分：**模型、接口和实现**。

同时我会以一些开源项目为案例，教你如何用这个方法去解读它们的设计，比如：

- 我们怎样理解Spring DI容器模型，它的模型怎样有效解决了其面对的问题；
- 如何理解Ruby on Rails的接口，我们可以从其接口设计中借鉴哪些内容；
- Kafka的实现有哪些独特之处，实现的诸多细节中，我们应该关注哪些内容。

通过对这些案例的解读，你会切实地感受到融会贯通的好处，真正做到快速了解一个软件的设计。

慢慢地，当你在业务和技能上有了一定的积累，你将有机会做属于自己的设计。你负责的工作内容也将会从一个小功能到一个完整的小业务，从到一个模块到一个系统。随着你的能力不断提升，你负责的内容会逐渐增多，复杂度逐步升级，对你设计能力的要求也会随之攀升。

这时，你就需要掌握一些软件设计的基础知识。我会把软件设计中最重要的部分交付给你，包括：

- 程序设计语言；
- 编程范式；
- 设计原则；
- 设计模式；
- 设计方法。

程序设计语言，是软件设计落地的基础。任何设计都依赖程序设计语言来实现。但任何语言也都有自己的局限，**我将带领你横跨语言学语言，让你不再局限于某一种语言，而是择其善者而从之，更好地落地你的设计。**

编程范式，是代码编写的风格，决定着你在设计时可以用到哪些元素：是模块、是对象，还是函数。在不同层次的设计中，选择不同的编程范式已经成为今天开发的主流。在这个主题下，我选择了几个最主流的编程范式，包括结构化编程、面向对象和函数式编程，帮你建立起软件设计的根基。

设计原则，是你在进入到具体设计的层面时，可以用来评判自己工作结果的一个衡量标准。我会给你介绍面向对象的主流设计原则：SOLID原则。一来面向对象是当今的主流开发方式，二来SOLID原则也是比较成体系的设计原则，它本身也在不断发展。

设计模式，是设计原则在具体场景下的应用。不过，这个话题展开之后，内容会非常多，而且有很多书和课程都讲到了，所以，我并不准备把它当作重点。但我会和你分享一些学习设计模式的心得，帮助你将设计模式的相关知识贯穿起来。

当你手里有了诸多工具之后，接下来就需要用这些工具去做自己的设计了。这就轮到**设计方法**登场了。

我会用领域驱动设计（也就是 DDD，Domain Driven Design）进行讲解，这是目前最为完整、有效的应对复杂业务场景的设计方法，包括了从如何识别概念到如何建立模型。在这个课程中，我准备将DDD的基础知识贯穿起来，做一个结构性的介绍。你会发现，有了前面知识的铺垫，DDD理解起来一点都不困难。

有了基础知识，在课程最后，我们还会在**巩固篇**中操练一下，将学到的软件知识应用起来。在这个模块中，我会结合自己的开源项目Moco，来讲讲如何设计一个程序库；还会借着一个数据采集的项目，谈谈如何构建起一个可扩展的模型。另外，因为大多数人在实际工作中面对的都是一个既有的项目，所以，我还会讲讲，如何对既有项目做设计上的改进。

01 软件设计到底是什么?
02 分离关注点：软件设计至关重要的第一步
03 可测试性：一个影响软件设计的重要因素

了解一个软件的设计

04 三步走：如何了解一个软件的设计?
05 Spring DI 容器：如何分析一个软件的模型?
06 Ruby on Rails：如何分析一个软件的接口?
07 Kafka：如何分析一个软件的实现?

设计一个软件

程序设计语言

08 语言的模型：如何打破单一语言局限，让设计更好落地?
09 语言的接口：语法和程序库，软件设计的发力点
10 语言的实现：运行时，软件设计的地基
11 DSL：你也可以设计一门自己的语言

编程范式

12 编程范式：明明写的是 Java，为什么有人说是 C 代码?
13 结构化编程：为什么做设计时仅有结构化编程是不够的?
14 面向对象之封装：怎样的封装才算是高内聚?
15 面向对象之继承：继承是代码复用的合理方式吗?
16 面向对象之多态：稀疏平常的多态，却是软件设计的大杀器?
17 函数式编程：不用函数式编程语言，怎么写函数式的程序?

17 函数式编程：不用函数式编程语言，怎么写函数式的程序？
18 函数式编程之组合性：函数式编程为什么如此吸引人？
19 函数式编程之不变性：怎样保证我的代码不会被别人破坏？

设计原则与模式

20 单一职责原则：你的模块到底为谁负责？
21 开放封闭原则：不改代码怎么写新功能？
22 Liskov 替换原则：用了继承，子类就设计对了吗？
23 接口隔离原则：接口里的方法，你都用得到吗？
24 依赖倒置原则：高层代码和底层代码，到底谁该依赖谁？
25 设计模式：每一种都是一个特定问题的解决方案
26 简单设计：难道一开始就要把设计做复杂吗？

设计方法

27 领域驱动设计：如何从零开始设计一个软件？
28 战略设计：如何划分系统的模块？
29 战术设计：如何构建一个可扩展的模型？

巩固篇

30 程序库的设计：Moco 是如何解决集成问题的？
31 应用的设计：如何设计一个数据采集平台？
32 应用的改进：如何改进我们的软件设计？
结束语 | 那些没有讲的事儿

写在最后

最后，再自我介绍一下。我叫郑晔，一个从业近二十年的程序员，《10x程序员工作法》专栏作者。很高兴又回到极客时间，和你分享我对软件设计的理解。

如果说《10x 程序员工作法》这门课是在告诉你要做正确的事，做有价值的需求，别把时间浪费在不该做的事情上，那《软件设计之美》这门课就是告诉你如何把事做对，如何建立有效的模型，划清模块之间的边界，所以，二者可谓一脉相承。

不想当将军的士兵不是一个好士兵，不想做设计的程序员不是一个好程序员。写程序的人谁不想操刀一个大型的系统，但不懂软件设计的人能摆弄的代码规模是有限的，而这也限定了一个人的成长高度。

学习软件设计，是让你的把控能力从一段代码扩展到一个模块，再扩展到一个子系统，再扩展到一个大系统的必备手艺，是程序员从“家常菜厨师”到“大厨”的进阶之路。

不过，你也不必把软件设计想象得过于高大上，很多设计理念既可以用来设计一个系统，也可以运用于日常开发之中，它就在你的身边。今天多学习一点设计，明天就能多发现一个问题。

如果你曾与我一样，走入过软件开发的迷途；如果你希望自己的软件开发能力再上层楼；或者你只是对软件设计充满好奇，那么，欢迎加入我的课程。也欢迎你把自己的现状和预期写在留言区，当课程结束时，让我们共同见证你的成长！

你准备好了吗？让我们正式开启软件设计之旅，一起领略软件设计的美妙！

精选留言：

- Kăfκă²⁰²⁰ 2020-05-25 20:10:00
  欢迎郑老师回来 [2赞]

- HackMSF 2020-05-26 01:06:53
  当我看到这段时眼前一亮，因为我当前就刚进入一家公司做BI，我入职两个星期发现我对当前规模的复杂度的把控能力是不足的，具体表现就是容易陷入细节，不知道该怎样搞懂系统的设计，我很期待。

  “大多数人在理解一个软件时，总会出现一个问题，就是眉毛胡子一把抓，直奔代码细节而去。这样不仅增加了我们的时间成本，还会迷失在细节之中，只见树木不见森林。所以在这个课程中，我会教你一个快速了解现有软件设计的方法，那就是抓住这个软件最核心的三个部分：模型、接口和实现。”

- Jxin 2020-05-25 22:35:05
  1.欢迎郑老师回来！！
  2.去年学习10x时，菜得很，受益良多。今年在谈软件设计，已有不少实践，希望能真正参与到专栏中来，而不再只是仰望。
  3.目前罗列的内容，知识面大了。相信郑老师主心骨应该还是串联这些个内容的骨干网络。但，细节与骨干实在不好把握，毕竟篇幅有限。有些许担忧。
  4.望再创佳绩。共同进步。

- Aliliin 2020-05-25 21:38:39
  昨天看到一个段子。问：你去年写了十万行代码，今年怎么写的这么少？
  答：因为我学会了使用 for 条件语句。😳

- Geek_bdd0e7 2020-05-25 19:12:20
  我一直很想学习一下在编程中如何从定义到属性到定理到模型再到模型间的关系，最后从模块构建一个系统。

  当中有令人兴奋的地方。

- Geek_46c5dd 2020-05-25 17:56:03
  需求的规模是增长的，当规模还不大的时候，需要用到设计吗？是不是crud就可以了？